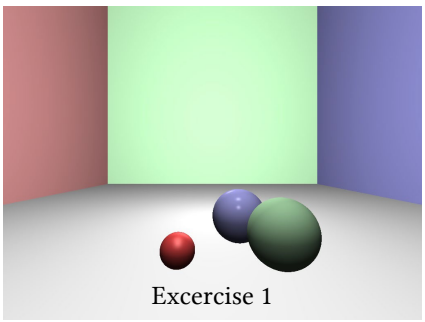
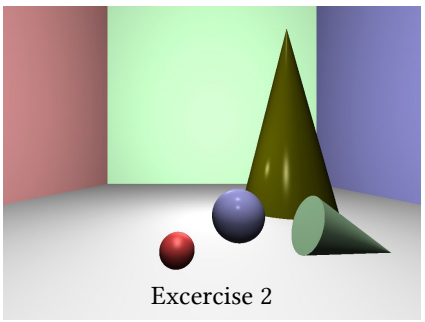
Università
della
Svizzera
italiana

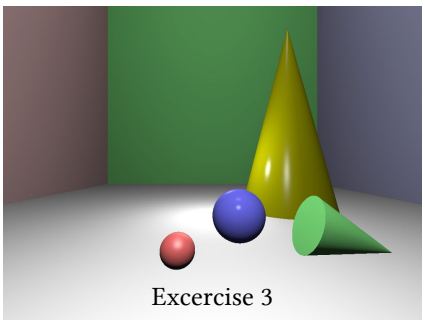
Faculty
of Informatics

Computer Graphics

Assignment 2: Transformations & Tone Mapping


Excercise 1


Excercise 2


Excercise 3

In this assignment, you will enhance our current raytracer by adding new primitives (plane and cone) and incorporating geometrical transformations that will allow you to define the final size and orientation of the cones. You will also improve the lighting handling by adding distance-based attenuation for lights and tone mapping to map the result of rendering to image values. You will be provided with an updated framework that clearly marks the places for implementing the required functionality. The updated framework is also a solution for the previous assignment. Upon completing the exercises, you should be able to generate images similar to the ones above.

Updated framework

We updated the framework as follow:

- The definition of the *Object* class contains now variables for three matrices:
  - `transformationMatrix` representing the transformation of the object in the global coordinate system,
  - `inverseTransformationMatrix`, the inverse of this transformation,
  - and `normalMatrix` for the normal vectors transformation in the global coordinate system;
- The *Object* class now also contains the template of the function `setTransformation` which should set the `transformationMatrix` and compute the other two matrices based on it;
- Empty definitions of the *Plane* and the *Cone* classes;
- Dummy function `toneMapping`.

The code is commented. Before continuing with the assignment, please familiarize yourself with the updates and check the code for comments indicating the places which should be modified. The current template is a solution for the previous codding assignment. Whether you want to start directly with the provided template or continue from your solution to the previous assignment, it is up to you. In the latter case, you are free to copy the above updates to the framework.

Useful functionality of the GLM library

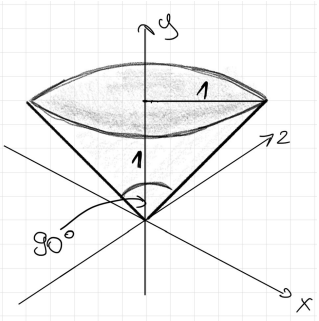
- `glm::mat4` type for 4×4 matrices;
- functions `glm::inverse` and `glm::transpose` allow you to easily compute the inverse and transpose of a given matrix,
- functions `glm::translate`, `glm::scale`, and `glm::rotate` enable an easy definition of translation, scaling, and rotation matrices;
- operation on matrices, vectors can be written as you would normally do on a paper, e.g., you can directly multiply matrix by matrix or matrix by vector, as long as the dimensions agree;
- you can easily extend a `vec3` by a homogeneous coordinate, multiply it by 4×4 and cast it back to `vec3`, e.g., :
  `glm::vec3 newOrigin = matrix * glm::vec4(origin, 1.0);` or
  `glm::vec3 newDirection = matrix * glm::vec4(direction, 0.0);`

Exercise 1 [5 points]

In this exercise, your task is to implement a plane-ray intersection routine and add six planes to the scene, forming a box around the spheres we already have. More specifically, the template already contains the class `Plane`. Your tasks is to implement the `intersection` function. Like what was discussed during the lecture, we define a plane using one point and the normal. Additionally, the constructor of the class can take a material structure as an argument. After implementing the intersection routine, please add to the scene six planes such that they form a box extending from -15 to 15 along x direction, from -3 to 27 along y direction, and from -0.01 to 30 along z direction. You can specify the material of the planes according to your preference, but they should not be completely black. Note, that for this exercise, you do not need to implement transformations, and therefore, you can assume that the planes are defined directly in the global coordinate system.

Exercise 2 [5 points]

Start by implementing the *Cone* class, such that it represents a simple cone as shown in the image on the right. The cone is **open**, it has 90 degrees opening angle, and the height equal 1. More formally, the equation of the cone is given by $x^2 + z^2 = y^2$ for $y \in [0, 1]$.



To be able to transform the cone you should:

- finish the implementation of the function `Object::setTransformation`,
- implement the intersection with the cone in the local coordinate system of the cone, i.e., before performing the intersection, transform the ray to the local coordinate system using transformation matrices, and then perform the intersection with the transformed ray. Afterwards, transform all the information to the global coordinate system as discussed during the lecture. The intersection point, normal vector, and the distance to the intersection have to be correctly defined in the final *Hit* structure so the rest of the code works correctly.

After implementing the ray-cone intersection routine, add two cones to the scene. To this end, modify the `sceneDefinition` function. Remember to set the transformation matrix for each cone using the function `setTransformation` before adding each cone to the collection of the objects.
You should add two cones as on the image above. The yellow cone should be highly specular. Its height is 12, and the radius of the bottom part is 3. The tip of the cone is at point $(5, 9, 14)$. The green diffuse cone has height 3 and radius at the base 1. The tip is located at point $(6, -3, 7)$, and it is rotated around the z-axis such that it lies with the side wall parallel to the ground. While you do not have to be very accurate to match the

material appearance from the image, please make the yellow cone highly specular such that the verification of the normal vectors and lighting computation is easier.
Finally, extend the implementation of the ray-cone intersection such that the cone is closed; i.e., add a disk that closes the cones. The image above visualizes the closed green cone. You can use the *Plane* class for creating the disk.

Exercise 3 [5 points]

Make your illumination more realistic. Introduce the attenuation of the light due distance in `PhongModel` function. You should also take care of the tone mapping and gamma correction by implementing a simple tone mapping routine. To make the image look nice, you will most likely need to tweak the intensities of the lights and coefficients in the material definition. Since we have not considered gamma correction before, our current raytracer has exaggerated ambient illumination. Try reducing both the intensity of the ambient light and the ambient coefficient for all materials. You are encouraged to play around with all the settings related to the light computation to create an image you like. However, please do not change the geometry of the scene, i.e., the positions and sizes of the objects.

Submission

Your submission **must** contain one ZIP-file with:

- a readme file or a PDF document file with information about which exercises you solved, the authors of the solutions, and the explanation of encountered problems, if any,
- an image file, *result.ppm*, containing the final image you could render,
- a directory named *code* containing all the source code used to generate the image.

The source code, upon compilation, should generate the image identical to the submitted *result.ppm* file. Your code should compile by calling *g++ main.cpp*. The ZIP file name must be of a form *surname1_surname2.zip* for a team of two, and *surname.zip* for a single submission. Only one person from the team should submit the solution.

Solutions must be submitted via iCorsi by the indicated there deadline.